FBSNG Application Programmer's Interface Reference Manual

Version 1.5

Krzysztof Genser, Tanya Levshina, Igor Mandrichenko

Farms and Clustered Systems Group
Fermi National Accelerator Laboratory

Table of Contents

1 Introduction

This document is a part of set of the FBSNG documentation. It describes functionality of Python version of FBSNG Application Programmer's Interface (API). Description of FBSNG features is left beyond the scope of this document. Other parts of documentation should be consulted for more information on specific FBSNG features:

- FBSNG User's Guide describes FBSNG command line interface as well as basic features;
- FBSNG Resources Concepts document describes the concepts of FBSNG resource management;
- FBSNG Scheduler document is about scheduling algorithms and related configuration issues;
- FBSNG Installation and Administration Guide describes installation and support procedures.

2 FBSNG API Overview

The FBSNG API provides access to FBSNG run-time and configuration information. It is organized as a set of classes divided into the following major groups:

- **API front-end FBSClient class.** This class performs all communication transactions between API client and FBSNG components. In order to use FBSNG API, user application must create an object of this class. An FBSClient object provides methods for:
  - Job submission, monitoring and control
  - Farm resource utilization monitoring
  - Queue monitoring and control
  - Monitoring and control of farm nodes
  - Obtaining farm configuration information
  - Dynamic modification of farm configuration

  FBSClient object creates objects of "informational" API classes such as FBSJobInfo, FBSSectionInfo, FBSProcessInfo, etc.
- **Job construction and submission:**
  - FBSJobDesc – representation of an FBSNG job to be submitted. Each FBSJobDesc object is a container of one or more FBSSectionDesc objects.
  - FBSSectionDesc – representation of individual FBSNG job section as a part of a job to be submitted to FBSNG.

  In order to submit a job using API, user must
  1. Create an FBSClient object
  2. Create FBSJobDesc object and populate it with one or more FBSSectionDesc objects either using existing Job Description File (see Appendix A) or creating FBSSectionDesc objects and adding them to the FBSJobDesc object.
  3. Use FBSClient object to send FBSJobDesc object for submission.

  For more information, refer to FBSJobDesc and FBSSectionDesc sections of this document.
- **Run-time batch job and farm status monitoring.** The following "informational" classes provide run-time information about batch jobs, farm resource utilization and FBSNG configuration.
  - FBSJobInfo – job monitoring and control;

  - FBSSectionInfo – monitoring and control of individual job section;
  - FBSProcessInfo and FBSSubProcessInfo – monitoring of individual batch processes and their subprocesses;
  - FBSQueueInfo – FBSNG queue monitoring, control and configuration;
  - FBSNodeClassInfo – farm node class configuration and monitoring;
  - FBSNodeInfo – individual farm node status monitoring and control;
  - FBSProcTypeInfo – configuration and run-time information about a process type.

  Objects of all informational classes are created by FBSClient object or objects of other FBSNG API classes. API client application should not create objects of these classes directly.
- **Asynchronous FBSNG event notification.** FBSEventListener class is a virtual base class that provides an interface to FBSNG Event Manager. Using this class, FBSNG API client application can build its own event-based job status monitoring agent.

Currently, only Python binding of FBSNG API is available.

Relationships between different API classes are shown on Fig. 1.

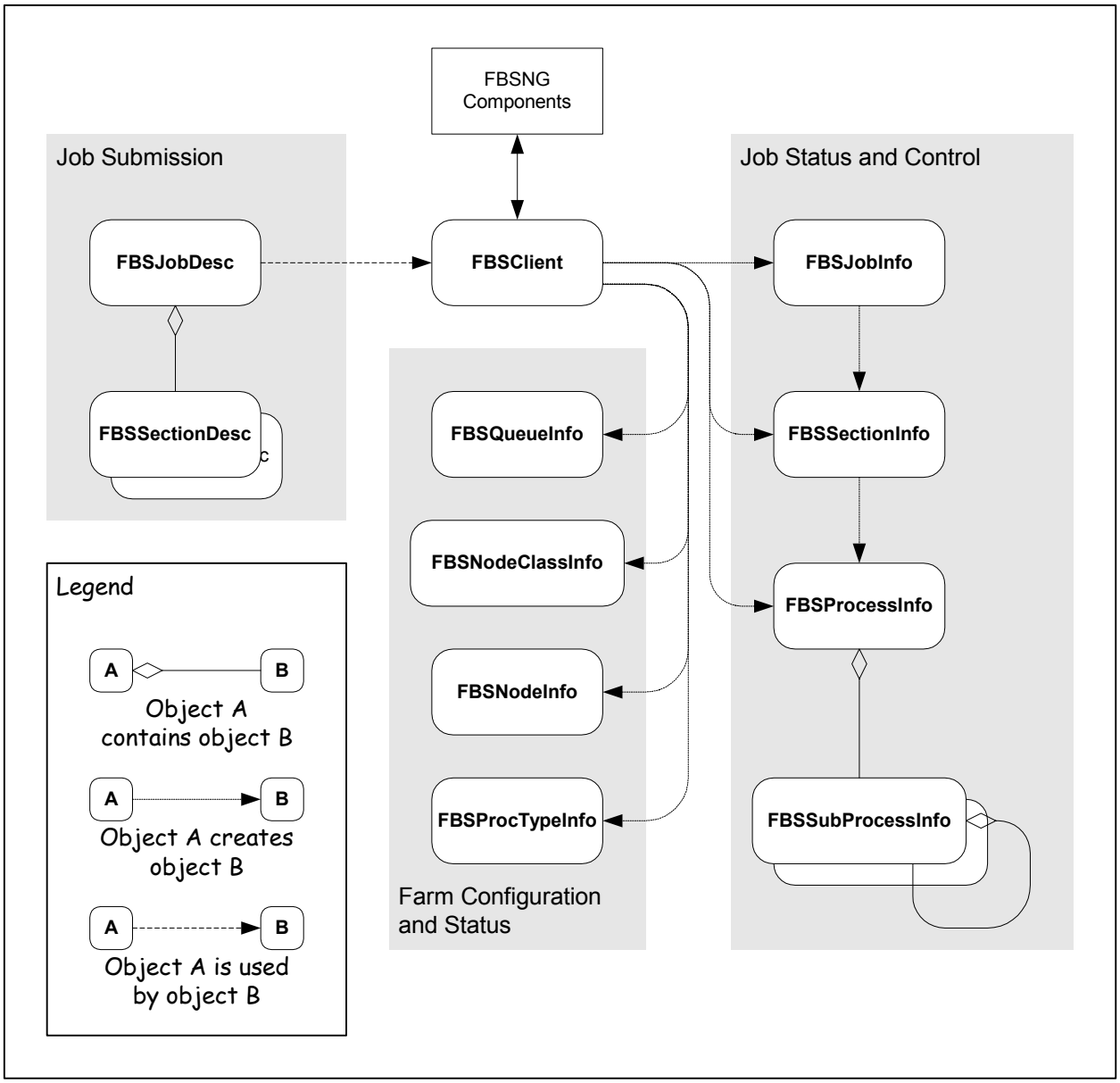

Figure 1. FBSNG API Classes

3 Using FBSNG API

In order to use FBSNG API, client application must import all or only necessary names from FBS_API module using one of the following methods:

```
from FBS_API import *          # import all names

from FBS_API import FBSClient, FBSJobDesc # import some names

import FBS_API                 # import only the module
```

If first method is used, the following names will be imported: FBSClient, FBSJobDesc, FBSSectionDesc and FBSError. If third method is used, class names will have to be explicitly qualified with module name "FBS_API". All examples of Python code in this document are written assuming first or second method was used.

Actual location of FBS_API.py module must be included into Python library search path defined by PYTHONPATH. FBSNG comes with necessary user environment set-up scripts that assign correct value to this and other environment variables. UPS users can use FBSNG API after issuing "setup fbsng" command. See FBSNG User's Guide for more details on setting up user environment.

4 Common Features of API Classes

4.1 Returning Status

Many API methods return completion status information as a 2-tuple (status, reason). By convention, status 1 indicates successful completion, and status 0 indicates failure. On success, the reason field is either 'OK' or some warning message. On failure, the reason field of the tuple contains a specific explanation of what happened.

```
For example:
      # Cancel or kill a section
      from FBS_API import FBSClient
      fc=FBSClient()
      si = fc.getSection('123.MAIN')
      sts, reason = si.kill()
      if sts:
            if reason == 'OK':
                  print 'Done'
            else:
                  print 'Warning: ', reason
      else:
            print 'Failed: ', reason
```

4.2 Generated Exceptions

Almost all methods of API classes described below generate standard Python KeyError exception on an attempt to get information or change state of a non-existing object. For example, every section normally finishes and eventually is removed from FBSNG. After the section is removed, refresh method of FBSSectionInfo class will generate a KeyError exception. The exception supplies the client with a textual explanation of the reason for generating the exception. The following is an example of proper exception handling for such situations:

```
# wait for section '123.MAIN' to finish and
# get removed from FBSNG

from FBS_API import *
fc=FBSClient()
si=fc.getSection('123.MAIN')
while si.State != 'done':
      time.sleep(10)
      try:  si.refresh()
      except KeyError, msg:
            print 'Section disappeared:', msg
            sys.exit(1)
print 'Section finished at ', time.ctime(si.ExitTime)

# now wait until is disappears
while 1:
      time.sleep(10)
      try:  si.refresh()
      except KeyError, msg
            print 'Section disappeared:', msg
            sys.exit(1)
```

Note that after a section finishes, it is kept in FBSNG for some time specifically to make sure that applications such as this have a chance to get exit status and other information about the section that just has finished.

Another possible type of exception generated by virtually any method of FBSNG is socket.error. This exception is generated when the BMGR process is not running, or is in initial recovery state, ignoring connection requests from the API clients. Normally, if an API client receives such an exception, the failed operation should be repeated after several seconds.

FBSError exception defined in FBS_API module is generated only when the API detects an error in the communication protocol with other FBSNG components. This exception should not normally be generated. When this happens, it is likely to be caused by an error in the API or another FBSNG component. This should be reported to FBSNG support group.

5 FBSClient Class

FBSClient class provides communication between an API client application and FBSNG components.

5.1 FBSClient Methods

Constructor FBSClient

Purpose: creates new FBSClient object. All communication with FBSNG goes through FBSClient object. In order to use FBSNG API, client application must create at least one FBSClient object.

Synopsis: `FBSClient()`

Arguments: none

Return value: new FBSClient object

submitJob

Purpose: submits a job to FBSNG.

Synopsis: `submitJob(job_desc)`

Arguments:
- job_desc: FBSJobDesc object

Return value: 2-tuple (status, text)
- status: Integer – is 1 on success and 0 on failure
- text: String – the job id on success, or textual explanation of the reason for failure

Exceptions: if the job is empty (contains no sections) or some section parameters are invalid, the method generates ValueError exception.

```
Example:
      from FBS_API import FBSClient
      fc=FBSClient()
      j = FBSJobDesc(jdf)
      sts, txt = fc.submitJob(j)
      if sts:
            print 'Job id = ', txt
      else:
            print 'Submit failed: ', txt
```

getJobList

Purpose: queries FBSNG for list of IDs of jobs matching certain criteria

Synopsis: `getJobList(username=None, uid=-1, gid=-1)`

Arguments:

- username: String or None. If not None, only jobs submitted by this user will be returned.
- uid: Integer. Numeric UNIX user ID. If specified, the list will contain IDs of jobs submitted by the specified user.
- gid: Integer. Numeric UNIX group ID. If specified, the list will contain IDs of jobs submitted by the members of specified group.

Return value: List of Strings. Each element of the list is a job ID. List can be empty.

Example:

```
from FBS_API import *
fc=FBSClient()
lst = fc.getJobList(uid = 12345)
print 'Jobs submitted by user #12345:', string.join(lst)
```

getJob

Purpose: queries FBSNG for iformation about the job specified with its job ID. The method returns the information as FBSJobInfo object.

Synopsis: `getJob(job_id)`

Arguments:

- Job_id: String – ID of the job

Return value: FBSJobInfo object

Exceptions: if the job is not found, the method generates KeyError exception.

```
Example:
      from FBS_API import FBSClient
      fc=FBSClient()
      try:  j = fc.getJob(jobid)
      except KeyError:
            print 'Job %s not found' % jobid
      else:
            print j.State
```

killJob

Purpose: kills or cancels user's job

Synopsis: `killJob(job_id, now=0)`

Arguments:

- job_id: String – ID of the job to kill or cancel
- now: Integer – if 1, sends SIGKILL signal to all job processes, otherwise, if 0 (default), or not specified, sends SIGINT, and then SIGKILL after grace period defined in FBSNG configuration.

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

In case when the job is not found, returns (0, <warning message>). Only the user who submitted the job (job owner) can kill it.

```
Examples:
      # kill or cancel all jobs submitted by the user
      from FBS_API import *
      fc=FBSClient()
      for jid in fc.getJobList(uid=os.getuid()):
            sts, reason = fc.killJob(jid)
            if sts:
                  print 'Job #%s killed' % jid
            else:
                  print 'Can not kill job #%s: %s' % (jid, reason)
```

getSection

Purpose: returns FBSSectionInfo with information about specific section by section ID.

Synopsis: `getSection(sect_id)`

Arguments:

- sect_id: String - ID of the section in the form: <job-id>.<section-name>. For example, "123.MAIN".

Return value: FBSSectionInfo object

Exceptions: if the specified section is not found, generates KeyError exception.

```
Example:
      from FBS_API import FBSClient
      fc=FBSClient()
      sid = '1234.MAIN'
      try:  si = fc.getSection(sid)
      except KeyError:
            print 'Section %s not found' % sid
      else:
            print si.State
```

getSectionOutput

Purpose: retrieves FBSNG internal copy of job section log by section ID.

Synopsis: `getSectionOutput(sect_id)`

Arguments:

- sect_id: String - ID of the section

Return value: list of tuples (time, message). Each element of the list contains one time-stamped message. Time is returned in the same format as from time.time(). Messages are text string which may contain one or more new-line characters.

Exceptions: if the section output file is not found, the method generates KeyError exception

```
Examples:
      from FBS_API import *
      fc=FBSClient()
      sid = '1234.MAIN'
      try:  log =  fc.getSectionOutput(sid)
      except KeyError:
            print 'Log for section %s not found' % sid
      else:
            for t, msg in log:
                  print time.ctime(t), ' ', msg
```

holdSection

Purpose: if the specified section is pending, holds it in the queue preventing it from starting.

Synopsis: `holdSection(sect_id, hold_time = -1)`

Arguments:

- sect_id: String – ID of the section to hold
- hold_time: Time or String – optional parameter. If specified, section will be automatically released after the hold_time. By default, section will be held until it is released. Hold time can be specified either as value returned from time.localtime(), or as text string in format described in "Hold Time Specification" section.

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the section is not found, the method generates KeyError exception.

Examples:

```
      # hold specific section till 1am tomorrow
      from FBS_API import FBSClient

      fc=FBSClient()
      try:
            sts, reason = fc.holdSection('123.ABC', '+1-01:00:00')
            if not sts:
                  print 'Can not hold the section: %s' % reason
      except:
            print 'Section not found'
```

releaseSection

Purpose: releases previously held section

Synopsis: `releaseSection(sect_id)`

Arguments:

- secti_id: String – ID of the section

Return value: 2-tuple (status, reason)
- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the section is not found, the method generates KeyError exception.

Examples:
```
# release all sections in a queue
from FBS_API import FBSClient
fc = FBSCLient()
for sid in fc.getQueue('LongQueue').Sections:
      sts, reason = fc.releaseSection(sid)
      if not sts:
            print 'Error releasing section %s: %s' % \
                        (sid, reason)
```

incSectPrio

Purpose: increments or decrements priority of a pending section

Synopsis: `incSectPrio(sect_id, increment)`

Arguments:
- sect_id: String – ID of the section
- increment: Integer – desired priority increment (if positive) or decrement (if negative)

Return value: 2-tuple (status, reason)
- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the section is not found, the method generates KeyError exception.

Example:
```
# increase priority of a section by 10
from FBS_API import FBSClient
fc=FBSCLient()
sts, reason = fc.incSectPrio('123.MAIN',10)
if not sts: print reason
```

killSection

Purpose: kills or cancels user's section. A section can be killed only by the user who submitted it.

Synopsis: `killSection(sect_id, now=0)`

Arguments:
- section_id: String – ID of the section to kill or cancel
- now: Integer = 0 – if 1, sends SIGKILL signal to all section processes, otherwise, if 0, or not specified, sends SIGINT, and then SIGKILL after grace period

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the section is not found, the method generates KeyError exception.

Examples:

```
# kill or cancel all sections in the queue
from FBS_API import *
fc=FBSClient()
for sid in fc.getQueue('LongQueue').Sections:
      sts, reason = fc.killSection(sid)
      if sts:
            print 'Section #%s killed' % sid
      else:
            print 'Can not kill section #%s: %s' % (sid, reason)
```

getProcess

Pupose: returns information about specific batch process identified by FBSNG process ID.

Synopsis: `getProcess(proc_id, local_details=1)`

Arguments:

- proc_id: String – Batch process ID (BPID) of the process in format: <job_id>.<section_name>.<logical_process_id>, for example "123.MAIN.2".
- local_details: Integer = 1 - optional argument, if local_details=1 (default), all available information about the process will be returned. Otherwise, if local_details=0, some returned information may be inaccurate, but the method will work faster. See description of FBSProcessInfo for details.

Return value: FBSProcessInfo object with information about the process

Exceptions: if the information about the process is not found, the method generates KeyError exception.

```
Examples:
      from FBS_API import FBSClient
      fc=FBSClient()
      pid = '1234.MAIN.5'
      try:  pi =  fc.getProcess(pid)
      except KeyError:
            print 'Process %s not found' % pid
      else:
            print 'Node %s, PID = %d' % (pi.Node, pi.UPID)
```

createQueue

Purpose: creates new queue. Queue is created in locked state, and must be unlocked later.

Synopsis: `createQueue(name, def_proc_type)`

Arguments:

- name: String – name of the queue to create
- def_proc_type: String – name of the default process type for the queue

Return value: FBSQueueInfo object representing the new queue

Exceptions: if a queue with the same name already exists, or the client is not authorized to perform the operation, or in case of another error generates ValueError exception with text string explaining the reason for failure as the value.

```
Example:
      # create new queue and then unlock it
      from FBS_API import *
      fc = FBSClient()
      try:  q = fc.createQueue('TestQ','MonteCarlo1')
      except ValueError, reason:
            print 'Can not create queue: ', reason
      else:
            q.unlock()
```

getQueueList

Purpose: returns list of FBSNG queues

Synopsis: `getQueueList()`

Arguments: none

Return value: List of Strings, each element is a queue name. List can be empty.

```
Examples:
      from FBS_API import *
      fc=FBSClient()
      qlst = fc.getQueueList()
      print "Queues: ",string.join(qlst)
```

getQueue

Purpose: returns information about specific queue by its name.

Synopsis: `getQueue(qname)`

Arguments:

- qname: String – name of the queue

Return value: FBSQueueInfo object with information about the queue

Exceptons: If the queue is not found, the method generates KeyError exception.

Examples:

```
from FBS_API import FBSClient
fc=FBSClient()
qlst = fc.getQueueList()
for qname in qlst:
      qi = fc.getQueue(qn)
      print '%s: %s' % (qn, string.join(qi.Sections))
```

holdQueue

Purpose: puts a queue on hold, disabling starting new sections from this queue

Synopsis: `holdQueue(qname)`

Arguments:

- qname: String – name of the queue

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the queue is not found, the method generates KeyError exception.

Examples:

```
# Hold individual queue
from FBS_API import FBSClient
fc = FBSClient()
sts, reason = fc.holdQueue('FastQ')
if not sts:
      print 'Error: ', reason
else:
      print 'done'
```

releaseQueue

Purpose: releases previously held queue, re-enabling starting of new sections from it.

Synopsis: `releaseQueue(qname)`

Arguments:

- qname: String – name of the queue

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the queue does not exist, the method generates KeyError exception.

```
Examples:
      # Release all existing queues
      from FBS_API import *
      fc = FBSClient()
      for qn in fc.getQueueList():
            sts, reason = fc.releaseQueue(qn)
            if sts:
                  print 'Queue <%s> released' % qn
            else:
                  print 'Can not release queue <%s>: %s' % (qn, reason)
```

lockQueue

Purpose: disables submitting new sections into the queue (locks the queue).

Synopsis: `lockQueue(qname)`

Arguments:

- qname: String – name of the queue

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the queue does not exist, the method generates KeyError exception.

```
Examples:
      # Lock all existing queues
      from FBS_API import *
      fc = FBSClient()
      for qn in fc.getQueueList():
            sts, reason = fc.lockQueue(qn)
            if sts:
                  print 'Queue <%s> locked' % qn
            else:
                  print 'Can not lock queue <%s>: %s' % (qn, reason)
```

unlockQueue

Purpose: unlocks previously locked queue.

Synopsis: `unlockQueue(qname)`

Arguments:

- qname: String – name of the queue

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the queue does not exist, the method generates KeyError exception.

```
Examples:
      # Unlock all locked queues
      from FBS_API import FBSClient
      fc = FBSClient()
      for qn in fc.getQueueList():
            if fc.getQueue(qn).IsLocked:
                  sts, reason = fc.unlockQueue(qn)
                  if sts:
                        print 'Queue <%s> unlocked successfully' % qn
                  else:
                        print 'Can not unlock queue <%s>: %s' % \
                              (qn, reason)
            else:
                  print 'Queue <%s> was not locked' % qn
```

removeQueue

Purpose: removes a queue from FBSNG configuration. Only empty queue can be removed.

Synopsis: `removeQueue(qname)`

Arguments:

- qname: String – name of the queue

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

```
Example:
      # remove queue "TestQueue"
      from FBS_API import FBSClient
      fc=FBSClient()
      sts, reason = fc.removeQueue("TestQueue")
      if sts:
            print 'Done'
      else:
            print reason
```

createGlobalResource

Purpose: creates new global resource.

Synopsis: `createGlobalResource(name, capacity)`

Arguments:

- name: String - name of the new resource. There should be no resource or resource pool of any type with the same name defined already.
- capacity: Integer – resource capacity

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String – the job id on success, or textual explanation of the reason for failure

```
Example:
      # create new global resource named 'NFS' with capacity 10
      from FBS_API import *
      print FBSClient.createGlobalResource('NFS',10)
      # will print either (1,'OK') on success or (0,reason)
      # on failure
```

setGlobalResource

Purpose: changes capacity for an existing global resource.

Synopsis: `setGlobalResource(name, capacity)`

Arguments:

- name: String - name of the new resource
- capacity: Integer – new resource capacity

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String – the job id on success, or textual explanation of the reason for failure

Exceptions: if the resource does not exist, generates KeyError exception.

```
Example:
      # set global resource 'NFS' capacity to 0 so that no
      # new job section that requires this resource can start
      from FBS_API import FBSClient
      print FBSClient.setGlobalResource('NFS',0)
      # will print either (1,'OK') on success or (0,reason)
      # on failure
```

getGlobalRsrcList

Purpose: returns list of known global resources and resource pools

Synopsis: `getGlobalRsrcList()`

Arguments: none

Return value: List of Strings, each element of the list is a resource name. List can be empty.

```
Examples:
      # print list of global resources including pools
      from FBS_API import FBSClient
      fc = FBSClient()
      list = fc.getGlobalRsrcList()
      print 'Global resources: ', string.join(list)
```

getGblRsrcQuota

Purpose: returns usage and quota information for a given global resource or a global resource pool and a process type

Synopsis: `getGblRsrcQuota(rsrc_name, proc_type)`

Arguments:
- rsrc_name: String – name of the global resource
- proc_type: String – name of the process type

Return value: 2-tuple (usage, quota)
- usage – current utilization of the resource by all processes and sections of the process type
- quota – process type utilization quota for this resources or None, if quota is unlimited.

Exceptions: if the resource does not exist or is not global, or the process type does not exist, the method generates KeyError exception.

```
Examples:
      from FBS_API import *
      usage, quota = FBSClient().getGblRsrcQuota('nfs_disk','Worker')
      if quota == None:
            quota = '(unlimited)'
      print 'Process type Worker uses %s of nfs_disk out of %s' %\
                  (usage, quota)
```

getGblRsrcUsage

Purpose: returns total usage and capacity for a global resource or a global resource pool on the farm.

Synopsis: `getGblRsrcUsage(rsrc_name)`

Arguments:
- rsrc_name: String – name of the global resource

Return value: 2-tuple (usage, capacity)
- usage – total amount of the resource currently allocated on the farm
- capacity – capacity for the resource

Exceptions: if the resource does not exist or is not global, the method generates KeyError exception.

```
Examples:
      from FBS_API import FBSClient
      usage, capty = FBSClient().getGblRsrcUsage('nfs_disk')
      print 'NFS disk usage: %d out of %d (%d%%)' % \
            (usage, capty, int(float(usage)/float(capty)*100.0))
```

removeGlobalResource

Purpose: removes a global resource from FBSNG configuration. A resource can be removed only if it is not in use.

Synopsis: `removeGlobalResource(name)`

Arguments:

- name: String – name of the global resource

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

```
Example:
      # remove all unused global resources
      from FBS_API import FBSClient
      fc=FBSClient()
      for rn in fc.getGlobalRsrcList ():
            usg, cap = fc.getGblRsrcUsage(rn)
            if not usage:
                  sts, reason = fc.removeGlobalResource(rn)
                  if sts:
                        print 'Resource %s removed' % rn
                  else:
                        print reason
```

createLocalResource

Purpose: creates new local resource.

Synopsis: `createLocalResource(name)`

Arguments:

- name: String - name of the new resource. There should be no resource or resource pool of any type with the same name defined already.

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String – the job id on success, or textual explanation of the reason for failure

```
Example:
      # create new local resource named 'disk1' for node class 'Worker'
      from FBS_API import *
      fc = FBSClient()
      sts, reason = fc.createLocalResource('disk1')
      if not sts:
            print reason
            sys.exit(1)
      nci = fc.getNodeClass('Worker')
      dct = nci.ResourceCap
      dct['disk1'] = 18       # 18 Gig
      sts, reason = nci.setRsrcCap(dct)
      if not sts:
            print reason
            sys.exit(1)
```

getLocalRsrcList

Purpose: returns list of known local resources and local resource pools

Synopsis: `getLocalRsrcList()`

Arguments: none

Return value: List of Strings, each element of the list is a resource name. List can be empty.

```
Examples:
      # print list of local resources including pools
      from FBS_API import FBSClient
      fc = FBSClient()
      list = fc.getLocalRsrcList()
      print 'Local resources: ', string.join(list)
```

getLclRsrcQuota

Purpose: returns usage and quota information for a given local resource or a local resource pool and process type.

Synopsis: `getLclRsrcQuota(rsrc_name, proc_type)`

Arguments:
- rsrc_name: String – name of the global resource
- proc_type: String – name of the process type

Return value: 2-tuple (usage, quota)
- usage – current utilization of the resource by all processes of the process type. If the resource is node attribute, usage is None.
- quota – process type utilization quota for this resources or None, if quota is not set up, or if the resource is a node attribute.

Exceptions: if the resource does not exist or is not local, or the process type does not exist, the method generates KeyError exception.

```
Examples:
      from FBS_API import FBSClient
      usage, quota = FBSClient().getLclRsrcQuota('cpu','Worker')
      if quota == None:
            quota = '(unlimited)'
      print 'Process type Worker uses %s units of CPU out of %s' %\
                  (usage, quota)
```

getLclRsrcUsage

Purpose: returns current usage and capacity for a local resource or a local resource pool on a farm node or on the whole farm. Usage for a pool is defined as sum of usages for all underlying resources.

Synopsis: `getLclRsrcUsage(rsrc_name, node_name = None)`

Arguments:
- rsrc_name: String – name of the local resource

- node_name: String or None – name of the node. If not specified, or is None, the method returns total farm utilization and capacity for the resource.

Return value: 2-tuple (usage, capacity)
If node_name is not None:

- usage – amount of the resource currently allocated on the node. If the resource is an attribute, usage is None.
- capacity – resource capacity of the node, or None if the resource is a node attribute or a pool composed of attributes.

If node_name is None or not specified:

- usage – total amount of the resource currently allocated on the whole farm, or None if the resource is an attribute.
- capacity – total resource capacity of the farm, or None if the resource is a node attribute or a pool composed of node attributes.

Exceptions: if the resource does not exist or is not local, or the node does not exist, the method generates KeyError exception.

```
Example:
      from FBS_API import *
      fc=FBSClient()
      usage, cap = fc.getLclRsrcUsage('dlt')
      print '%d out of %d DLT drives are in use on the farm' % \
            (usage, cap)
      uh, ch = fc.getLclRsrcUsage('dlt','fnsfh')
      print 'FNSFH has %d drives, %d available' % (ch, ch-uh)
```

removeLocalResource

Purpose: removes a local resource from FBSNG configuration. A resource can be removed only if it is not in use.

Synopsis: `removeLocalResource(name)`

Arguments:

- name: String – name of the local resource

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

```
Example:
      # remove all unused local resources
      from FBS_API import FBSClient
      fc=FBSClient()
      for rn in fc.getLocalRsrcList ():
            usg, cap = fc.getLclRsrcUsage(rn)
            if not usage:
                  sts, reason = fc.removeLocalResource(rn)
                  if sts:
                        print 'Resource %s removed' % rn
                  else:
                        print reason
```

createRsrcPool

Purpose: creates new resource pool.

Synopsis: `createRsrcPool(name, rsrc_list)`

Arguments:

- name: String – name of the pool to create. There should be no resource or resource pool of any type with the same name defined already.
- rsrc_list: List of Strings – list of the underlying resources to be combined in the pool. All listed resources must be of the same type: global or local or local attribute. Resource pools may not consist of other resource pools.

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String – the job id on success, or textual explanation of the reason for failure

```
Example:
      # create new resource pool combining attributes representing
      # different versions of Linux
      from FBS_API import FBSClient
      sts, reason = FBSClient().createRsrcPool('Linux',
                        ['Linux5','Linux6'])
      if sts:
            print 'OK'
      else:
            print 'Error: ', reason
```

Exceptions: if one of the underlying resources does not exist, generates KeyError exception.

setRsrcPool

Purpose: changes composition of an existing resource pool.

Synopsis: `setRsrcPool(name, rsrc_list)`

Arguments:

- name: String – name of the pool.

- rsrc_list: List of Strings – list of the underlying resources to be combined in the pool. All listed resources must be of the same type: global or local or local attribute. Resource pools may not consist of other resource pools.

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String – the job id on success, or textual explanation of the reason for failure

Exceptions: if the specified resource pool or one of the listed underlying resources does not exist, generates KeyError exception.

```
Example:
      # remove resource 'disk1' from all existing local resource pools
      from FBS_API import *
      fc = FBSClient()
      pools = fc.getLocalPoolList()
      for pool in pools:
            lst = fc.getResourcePool(pool)
            if 'disk1' in lst:
                  lst.remove('disk1')
                  sts, reason = fc.setRsrcPool(pool, lst)
                  if not sts:
                        print 'Can not remove disk1 from %s: %s' % \
                              (pool, reason)
```

getLocalPoolList

Purpose: returns list of local resource pool names.

Synopsis: `getLocalPoolList()`

Arguments: none

Return value: List of Strings, each element is a name of a local resource pool. The list can be empty.

```
Examples:
      # print composition of all local resource pools
      from FBS_API import FBSClient
      fc = FBSClient()
      for pool_name in fc.getLocalPoolList():
            print 'Pool %s:',
                  string.join(fc.getResourcePool(pool_name))
```

getGlobalPoolList

Purpose: returns list of global resource pool names.

Synopsis: `getGlobalPoolList()`

Arguments: none

Return value: List of Strings, each element is a name of a global resource pool. The list can be empty.

```
Examples:
      # print composition of all global resource pools
      from FBS_API import *
      fc = FBSClient()
      for pool_name in fc.getGlobalPoolList():
            print 'Pool %s:',
                  string.join(fc.getResourcePool(pool_name))
```

getResourcePool

Purpose: returns list of names of the underlying resources the resource pool cosists of.

Synopsis: `getResourcePool()`

Arguments: none

Return value: List of Strings, each element is a name of an underlying resource. The list can be empty.

Exceptions: if the resource pool does not exist, the method generates KeyError exception.

```
Examples:
      # print composition of all resource pools
      from FBS_API import FBSClient
      fc = FBSClient()
      lst = fc.getLocalPoolList() + fc.getGlobalPoolList()
      for pool_name in lst:
            print 'Pool %s:',
                  string.join(fc.getResourcePool(pool_name))
```

removeResourcePool

Purpose: removes a resource from FBSNG configuration. A resource pool can be removed only if the pool is not in use.

Synopsis: `removeResourcePool(name)`

Arguments:

- name: String – name of the resource pool

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

```
Example:
      # remove all unused node attribute pools
      from FBS_API import FBSClient
      fc=FBSClient()
      for rp in fc.getLocalPoolList():
            usg, cap = fc.getLclRsrcUsage(rn)
            if not usage and cap == None:
                  sts, reason = fc.removeResourcePool(rp)
                  if sts:
                        print 'Pool %s removed' % rp
                  else:
                        print reason
```

createNodeClass

Purpose: creates new node class without any nodes assigned to it. FBSNodeClassInfo methods should be used to add configure node class resources and add nodes to it.

Synopsis: `createNodeClass(name)`

Arguments:

- name: String – name of the node class to create

Return value: FBSNodeClassInfo object representing the new node class.

Exceptions: if a node class with the same name already exists, or the client is not authorized to perform the operation, or in case of another error generates ValueError exception with text string explaining the reason for failure as the value.

```
Example:
      # create new node class, set resources for it,
      # and add 10 new nodes
      from FBS_API import FBSClient
      fc = FBSClient()
      try:  nc = fc.createNodeClass('MCWorkers')
      except ValueError, reason:
            print 'Can not create node class: ', reason
      else:
            nc.setRsrcCap({'cpu':200,'MCWorker':None)
            for i in range(10):
                  nc.addNode('pc%d' % (i+1))
```

getNodeClassList

Purpose: returns list of configured node classes.

Synopsis: `getNodeClassList()`

Arguments: none

Return value: list of Strings, each item is node class name.

```
Examples:
      from FBS_API import FBSClient
      lst = FBSClient().getNodeClassList()
      print 'Node classes are: ', string.join(lst)
```

getNodeClass

Purpose: returns information on given node class.

Synopsis: `getNodeClass(cname)`

Arguments:

- cname: String – name of the node class

Return value: FBSNodeClassInfo object with information on the node class

Exceptions: if the node class does not exist, the method generates KeyError exception.

```
Examples:
      from FBS_API import *
      nci = FBSClient().getNodeClassInfo('IO_Class')
      print 'IO Nodes: ', string.join(nci.Nodes)
      print 'Resources: '
      for rn, rc in nci.ResourceCap.items():
            print '%s: %d' % (rn, rc)
```

removeNodeClass

Purpose: removes a node class from FBSNG configuration. A node class can be removed only if there are no nodes of this class.

Synopsis: `removeNodeClass(name)`

Arguments:

- name: String – name of the node class

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

```
Example:
      # remove all empty node classes
      from FBS_API import FBSClient
      fc=FBSClient()
      for ncn in fc.getNodeClassList():
            nc = fc.getNodeClass(ncn)
            if not nc.Nodes:
                  sts, reason = fc.removeNodeClass(ncn)
                  if sts:
                        print 'Node class %s removed' % ncn
                  else:
                        print reason
```

getNode

Purpose: returns information about individual farm node.

Synopsis: `getNode(name)`

Arguments:

- name: String – name of the node

Return value: FBSNodeInfo object with information about the node

Exceptions: if the node does not exist, the method generates KeyError exception.

```
Examples:
      from FBS_API import FBSClient
      fc = FBSClient()
      nci = fc.getNodeClass('Worker')
      print 'Nodes of class Worker':
      for nn in nci.Nodes:
            ni = fc.getNode(nn)
            print '%s up:%d held:%d ' % (nn, ni.IsUp, ni.IsHeld),
            if ni.IsHeld:
                  print 'reason: ', ni.HoldReason
            else:
                  print ''
```

holdNode

Purpose: puts a farm node on hold: prevents new batch processes from starting on the node. This operation does not affect processes already running on the node.

Synopsis: `holdNode(name, reason)`

Arguments:

- name: String – name of the node to hold
- reason: String – reason for holding

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the node does not exist, the method generates KeyError exception.

```
Example:
      # hold all nodes of class WorkerA for maintenance
      from FBS_API import FBSClient
      fc = FBSClient()
      for nn in fc.getNodeClass('WorkerA').Nodes:
            sts, reason = fc.holdNode(nn, "Held for maintenance")
            if not sts:
                  print 'Can not hold node <%s>: %s' % (nn, reason)
```

releaseNode

Purpose: releases previously held node

Synopsis: `releaseNode(name)`

Arguments:

- name: String – name of the node to release

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the node does not exist, the method generates KeyError exception.

```
Examples:
      # release all held nodes of class WorkerA
      from FBS_API import *
      fc = FBSClient()
      for nn in fc.getNodeClass('WorkerA').Nodes:
            if fc.getNode(nn).IsHeld:
                  fc.releaseNode(nn)
```

getProcessTypeList

Purpose: returns list of known process type names

Synopsis: `getProcessTypeList()`

Arguments: none

Return value: list of Strings, each item is name of a process type

Examples:

```
      from FBS_API import FBSClient
      lst = FBSClient().getProcessTypeList()
      print 'Process types: ', string.join(lst)
```

createProcessType

Purpose: creates new process type. Process type is created with unlimited quotas and zero default process requirements. These parameters must be set using appropriate methods of FBSProcTypeInfo class.

Synopsis: `createProcessType(name)`

Arguments:
- name: String – name of the process type to create

Return value: FBSProcTypeInfo object representing the new process type

Exceptions: if a process type with the same name already exists, or the client is not authorized to perform the operation, or in case of another error generates ValueError exception with text string explaining the reason for failure as the value.

```
Example:
      # create new process type, set default resource requirements
      # and quotas
      from FBS_API import *
      fc = FBSClient()
      try:  pt = fc.createProcessType('NewWorker')
      except ValueError, reason:
            print 'Can not create process type: ', reason
      else:
            pt.setProcRsrcDefs({'cpu':100,'Linux':None})
            pt.setRsrcQuota({'cpu':10000})
```

getProcessType

Purpose: returns information about a process type

Synopsis: `getProcessType(pt_name)`

Arguments:
- pt_name: String

Return value: FBSProcTypeInfo object

Exceptions: if the process type does not exist, the method generates KeyError exception.

`Example:` see examples for FBSProcTypeInfo class methods

removeProcessType

Purpose: removes a process type from the FBSNG configuration. A process type can be removed only if there are no sections in any state using this process type.

Synopsis: `removeProcessType(name)`

Arguments:
- name: String – name of the process type

Return value: 2-tuple (status, reason)
- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

```
Example:
      # remove all process types with names starting with Z
      from FBS_API import FBSClient
      fc=FBSClient()
      for ptn in fc.getProcessTypeList():
            if ptn[0] == 'Z':
                  sts, reason = fc.removeProcessType(ncn)
                  if sts:
                        print 'Process type %ptn removed' % ncn
                  else:
                        print reason
```

setTimeOut

Purpose: sets new value for FBSClient communication time-out

Synopsis: `setTimeOut(timeout)`

Arguments:

- timeout: Integer = -1 – communication time-out length in seconds. –1 means that FBSClient will re-try to connect to FBSNG indefinitely. This is optional argument. –1 is default.

Return value: previously set time-out value

```
Example:
      # Submit a job. If FBSNG is not running, exit after
      # 60 seconds
      from FBS_API import *
      import socket

      job = FBSJobDesc('my.jdf')
      fc=FBSClient()
      fc.setTimeOut(60)
      try:  sts, text = fc.submitJob(job)
      except socket.error:
            print 'FBSNG is not running, try again later'
      else:
            if sts:
                  print 'Job %s submitted' % text
            else:
                  print 'Can not submit the job: ', txt
```

6 FBSJobDesc Class

FBSJobDesc class is a job description data structure. It is used to build a job to be submitted. FBSJobDesc is a container of one or more FBSSectionDesc objects.

6.1 Methods

Constructor

Purpose: creates new FBSJobDesc object. Unless JDF is specified as the constructor's argument, the created object is empty (contains no sections).

Synopsis: `FBSJobDesc(jdf = None)`

Arguments:

- jdf: String – path to JDF. JDF will be read and newly created FBSJobDesc object will be populated with sections described in the JDF. See Appendix A for JDF format description. If unspecified, job description will contain no sections and will have to be populated using addSection method (see below).

Return value: newly created FBSJobDesc object

Exceptions: if JDF file contains syntax or another error, the method generates SyntaxError exception. In this case, exception value is a 3-tuple:

- Integer line number
- Line text (String)
- Explanation (String)

Examples:

```
# submit a job using JDF
from FBS_API import FBSClient, FBSJobDesc
fc = FBSClient()
try:  job = FBSJobDesc(sys.argv[1])
except SyntaxError, (lno, line, what):
      print 'Error in JDF %s at line %d: %s' % \
            (sys.argv[1], lno, what)
      print 'Line: [%s]' % line
sts, msg = fc.submitJob(job)
if sts:
      print 'Job id = ', msg
else:
      print 'Job submission failed: ', msg
```

getSection

Purpose: returns description of individual section of the job.

Synopsis: `getSection(sname)`

Arguments:

- sname: String – name of the section

Return value: FBSSectionDesc object with description of the section

Exceptions: if the job description does not contain specified section, the method generates KeyError exception.

Examples:

```
# print all sections of the job
from FBS_API import FBSJobDesc
job = FBSJobDesc('my.jdf')
for sn in job.sections():
      print job.getSection(sn)
```

addSection

Purpose: adds new section to the job description.

Synopsis: `addSection(section)`

Arguments:

- section: FBSSectionDesc object – description of the new section

Return value: none

Exceptions: if the job description already has section with the same name, the method generates ValueError exception

Examples:

```
# read job description template and replicate
# section 'WORK_1' 3 times
from FBS_API import *
fc = FBSClient()
job = FBSJobDesc('my.jdf')
s = job.getSection('WORK_1')  # get template section description
for i in range(3):
      s1 = s.clone('WORK_%d' % i+2) # replicate template section,
                                    # give it new name
      job.addSection(s1)            # add replicated section
```

hasSection

Purpose: queries whether the job description contains section with specified name

Synopsis: `hasSection(sname)`

Arguments:

- sname: String – name of the section

Return value: Integer – 1 if section with specified name exists, 0 otherwise.

Examples:

```
# add clean-up section from another file if it does
# not exist in JDF template
from FBS_API import FBSJobDesc
jtemp = FBSJobDesc('template.jdf')
if not jtemp.hasSection('Cleanup'):
      jclean = FBSJobDesc('clean-up.jdf')
      jtemp.addSection(jclean.getSection('Cleanup')
```

sections

Purpose: returns names of sections contained in the job description

Synopsis: `sections()`

Arguments: none

Return value: List of Strings – list of section names. The list can be empty.

Examples:

```
# merge descriptions from 2 JDFs
from FBS_API import *
job = FBSJobDesc()
job1 = FBSJobDesc('head.jdf')
job2 = FBSJobDesc('tail.jdf')
for sn in job1.sections():
      job.addSection(job1.getSection(sn))
for sn in job2.sections():
      job.addSection(job2.getSection(sn))
```

validateDependencies

Purpose: validates inter-section dependencies.

Synopsis: `validateDependencies()`

__repr__

Purpose: returns string representation of the job in JDF format. This method is implicitly called by Python print statement.

Synopsis:
```
repr(job_object)
job.__repr__()
```

Arguments: none

Return value: String with text representation of the job description

Examples:

```
      # add clean-up section from another file if it
      # does not exist in the template
      # then print and save as new JDF
      from FBS_API import FBSJobDesc
      jtemp = FBSJobDesc ('template.jdf')
      if not jtemp.hasSection('Cleanup'):
            jclean = FBSJobDesc ('clean-up.jdf')
            jtemp.addSection (jclean.getSection('Cleanup')
      print 'New JDF:'
      print jtemp
      f = open('new.jdf','w')
      f.write(repr(jtemp))
      f.close()
```

Arguments: none

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

```
Examples:
      # Read JDF and validate dependencies
      from FBS_API import *
      job = FBSJobDesc('my.jdf')
      sts, reason = job.validateDependencies()
      if not sts: print reason
```

7 FBSSectionDesc Class

FBSSectionDesc class contains description of individual job section. Objects of this class should be used to construct a job from sections by populating FBSJobDesc object.

7.1 Data Members

FBSSectionDesc object contains complete job section description. The following data members should be used to describe the section:

FBSSectionDesc Data Members

Name	Type	Default	Description
Name	String	(required)	Section name
ProcType	String	Queue default	Process type
HoldTime	Date/time or String	None	Hold-until time, or None, or -1 for hold-forever. String representation of the date/time is also acceptable.
PerProcRsrc	Dict	{}	Local and global resources consumed by each process
PerSectRsrc	Dict	{}	Global resources consumed by the section
Queue	String	(required)	Queue name
NProc	Int	1	Number of processes
Nice	Int	0	Run-time nice parameter
PrioInc	Int	0	Initial section priority relative to normal initial priority.
Need	0/1	0	NEED field
LeaderOnly	0/1	0	LEADER_ONLY field
Exec	List of Strings	(required)	User command
MailTo	String	None	E-mail address for section log
Depend	String	None	Section dependencies specification
SectOutput	String	None	Template for section log output file name
Stderr	String	FBS_%j.%n.out	Template for process stderr file names
Stdout	String	FBS_%j.%n.err	Template for process stdout file names
OnNodes	List of Strings	None (= on all available nodes)	Lists nodes processes of the section are allowed to start on

Exec field of FBSSectionDesc structure represents the command to be executed by each process of the section. The command can be specified either as a string or as list of strings. List of strings should be used if some words of the command contain spaces or other special characters.

For example, the following two fragments of code produce the same result:

```
s = FBSSectionDesc('run')
s.Exec = "/bin/echo Hello"

…
s.Exec = ["/bin/echo","Hello"]
```

In more complicated cases like the following, it is necessary to represent the command as a list:

```
s.Exec = ["su","theuser","-c","/bin/echo Hello"]
```

7.2 Methods

Constructor

Purpose: creates a new FBSSectionDesc object. The constructor can be used to set section parameters. In addition, the parameters can be modified later using direct assignment of new values to the object fields.

Synopsis: `FBSSectionDesc(name, override={}, keyword=value, …)`

Arguments:

- name: String – section name. Note that no two sections of the same job should have the same name.
- override: Dictionary = {} – optional parameter which can be used to assign values to parameters at construction time. Value of this parameter must be a dictionary with class field names as string keys and values as values.
- keyword arguments. Each keyword=value pair will be used to assign specified value to corresponding data member. Only class data member names are accepted as keywords.

Using override dictionary and keyword arguments produces the same results. These two methods can be used interchangeably or even in the same call to the constructor. If the same parameter is present both in override dictionary and as keyword argument, value of the later will be assigned. User can change any field value later assigning values directly to object fields.

Return value: newly created FBSSectionDesc object

```
Examples:
      # Create 4 exact copies of a section using different methods.
      # First, using only keyword arguments
      from FBS_API import *
      sd1 = FBSSectionDesc('WORK',  # section name
            Exec='/bin/sleep 10',   # Exec field
            Need=1,                 # Need field
            Nice=5,                 # Nice field
            Queue = 'WorkerQueue',  # Queue field
            LeaderOnly = 1,         # LeaderOnly flag
            NProc = 5)              # Number of processes

      # Second, using only override dictionary.
      # Note that field names are represented as strings.
      sd2 = FBSSectionDesc ('WORK', # section name
            override = {
                  'Exec':'/bin/sleep 10', # Exec field
                  'Need':1,               # Need field
                  'Nice':5,               # Nice field
                  'Queue':'WorkerQueue',  # Queue field
                  'LeaderOnly':1,         # LeaderOnly flag
                  'NProc':5}              # Number of processes
            )

      # Third, using both methods.
      sd3 = FBSSectionDesc ('WORK',       # section name
            override = {                  # override dictionary
                  'Nice':5,               # Nice field
                  'Queue':'WorkerQueue',  # Queue field
                  'LeaderOnly':1,         # LeaderOnly flag
                  'Need':0
                  },
                                    # keyword arguments:
            Exec='/bin/sleep 10',   # Exec field
            Need=1,                 # Need field. Note that this field
                                    # is mentined in override
                                    # dictionary too. In this case,
                                    # Need will be set to 1.
            NProc = 5)              # Number of processes

      # and now create another copy accessing object data
      # members directly
      sd4 = FBSSectionDesc ('WORK') # Set only section name
                                    # then set each field directly
      sd4.Exec='/bin/sleep 10'
      sd4.Need=1
      sd4.Nice=5
      sd4.Queue='WorkerQueue'
      sd4.LeaderOnly=1
      sd4.NProc=5
```

clone

Purpose: make copy of existing section description, optionally modifying some parameters.

Synopsis: `clone(name, override={}, keyword=value, …)`

Arguments:

- name: String – section name for the copy
- override: Dictionary = {} - same as for constructor
- keyword parameters, same as for constructor

This method has the same arguments with the same meaning as the constructor.

Return value: newly created section description object

Examples:

```
# Read JDF and replicate WORK_A section 7 times modifying
# some parameters
from FBS_API import *
job = FBSJobDesc('template.jdf')
s = job.getSection('WORK_A')
for suffix in 'BCDEFGH':
      s1 = s.clone('WORK_%s' % suffix,    # change name
                                          # and Exec field
            Exec = s.Exec + (' /scratch/%s.dat' % suffix))
      job.addSection(s1)            # add new section to the job
s.Exec = s.Exec + ' /scratch/A.dat'
# submit the job
print FBSClient().submitJob(job)
```

__repr__

Purpose: produces text representation of the section in JDF format. This method is implicitly used by Python 'print' statement.

Synopsis:
```
repr(section)
section.__repr__()
```

Arguments: none

Return value: text representation of the section.

Examples:

```
# Create a section description and print it
from FBS_API import FBSSectionDesc
s = FBSSectionDesc('Start', Queue='LongQ',
      Exec='/bin/process-data.sh XYZ123',
      NProc=5,
      PerProcRsrc={'scratch':5, 'tape':1})
print s
```

this fragment of code will print:

```
SECTION Start
EXEC = /bin/process-data.sh XYZ123
NUMPROC = 5
PROC_RESOURCES = scratch:5 tape:1
```

8 FBSJobInfo Class

Objects of this class represent information about batch jobs returned from FBSNG. They are generated by getJob method of FBSClient class. API users should not create FBSJobInfo objects. That is why constructor for this class is not documented.

8.1 Data Members

FBSJobInfo class has data members listed in the following table. All data members are "read-only" members. Any modifications of their values will be ignored.

FBSJobInfo Data Members

Name	Type	Description
ID	String	Job ID
UID	Int	UNIX user ID of the job owner
GID	Int	UNIX group ID of the job owner
State	String	Job state: "active" or "done"
Username	String	UNIX user name

8.2 Methods

sections

Purpose: returns list of names of job sections

Synopsis: `sections()`

Arguments: none

Return value: List of Strings – list of job section names

Examples: see example for getSection

getSection

Purpose: returns information about individual job section.

Synopsis: `getSection(sname)`

Arguments:

- sname: String – section name

Return value: FBSSectionInfo object with information about the section

Exceptions: if the job does not have specified section, or the job itself no longer exists, the method generates KeyError exception.

```
Examples:
      # List IDs of all sections of all jobs submitted by the user
      from FBS_API import FBSClient
      fc=FBSClient()
      for jid in fc.getJobList(uid=os.getuid()):
            j = fc.getJob(jid)
            for sn in j.sections():
                  s = j.getSection(sn)
                  print s.ID
```

kill

Purpose: sends request to kill the job, if the job is running, or cancel, if it is still pending. Only job owner can kill the job.

Synopsis: `kill(now=0)`

Arguments:

- now: Integer - if now=1, all processes of the job will be terminated immediately with SIGKILL signal. Otherwise, by default, SIGINT will be sent to all processes, and then, after grace period of time, SIGKILL will be sent.

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the job no longer exists, the method generates KeyError exception.

```
Examples:
      # Kill all jobs owned by members of the user's group:
      from FBS_API import FBSClient
      fc = FBSClient()
      for jid in fc.getJobList(gid=os.getgid()):
            sts, reason = fc.getJob(jid).kill()
            if not sts:
                  print 'Can not kill job #%s: %s' % (jid, reason)
```

refresh

Purpose: update an object's data members with current information

Synopsis: `refresh()`

Arguments: none

Return value: none

Exceptions: if the job no longer exists, the method generates KeyError exception.

Examples:

```
      # Wait for completion of a job
      from FBS_API import *
      fc=FBSClient()
      j=fc.getJob('1234')
      while j.State == 'active':
            time.sleep(10)    # sleep, and then refresh
            try:  j.refresh()
            except KeyError:
                  break       # job disappeared, it's done
```

9 FBSSectionInfo Class

Objects of FBSSectionInfo class contain information about FBSNG job sections. They are created by other FBSNG API objects such as FBSClient and FBSJobInfo. API clients should not create objects of FBSSectionInfo class. That is why its constructor is not documented.

9.1 Data Members

Data members of FBSSectionInfo class are listed in the table below. All data members are "read-only" in the sense that any modifications of their values will be ignored.

FBSSectionInfo Data Members

Name	Type	Description
ID	String	Section ID
JobID	String	Job ID
Name	String	Section name
ProcType	String	Process type
SubTime	Date/time	Submission date/time
HoldTime	Date/time	Hold-until time, or None, or -1 for hold-forever
StartTime	Date/time	Start time, or None
EndTime	Date/time	End time, or None
PerProcRsrc	Dict	Local and global resources consumed by each process
PerSectRsrc	Dict	Global resources consumed by the section
RsrcPoolDict	Dict	Dictionary translating requested resource pool names into names of actually allocated resources
Queue	String	Queue name
Prio	Int	Section priority. Modified by Scheduler.
UID	Int	User id
GID	Int	Group id
Username	String	Username
NProc	Int	Number of processes
ProcStats	Dictionary Int -> String	The dictionary is indexed with process number (1…NProc) and maps logical process id to process' state ("running","exited","pending")
Nice	Int	Run-time nice parameter
PrioInc	Int	Section scheduling priority increase (decrease) accumulated so far. See incPrio method
Exec	String	User command
Need	0/1	NEED parameter
LeaderOnly	0/1	LEADER_ONLY parameter
CPUTimeLimit	Int	CPU time limit in seconds
RealTimeLimit	Int	Real time limit in seconds
SectOutput	String	Template for section log output
Stderr	String	Template for process stderr files

Name	Type	Description
Stdout	String	Template for process stdout files
ExitCode	Int	Exit status for exited sections, otherwise None
Depend	String	Dependencies
JDFSeq	Int	The number of the section in JDF file, or sequence number of call to FBSJobDesc.addSection() the section was added to the job.
State	String	Section state (waiting, ready, running, done, exited, canceled, zombie)
OnNodes	List of Strings or None	List of nodes the section is allowed to run on, or None if no restrictions.

9.2 Methods

getProcess

Purpose: returns information about individual section process.

Synopsis: `getProcess(proc_no, local_details = 1)`

Arguments:

- proc_no: Integer – logical process ID
- local_details: Integer - optional argument, if local_details is 1 (default), all available information about the process will be returned. Otherwise, if local_details is 0, some returned information may be inaccurate or not up-to-date, but the method will work faster. See description of FBSProcessInfo (page 51) for more information.

Return value: FBSProcessInfo object with information about the process

Exceptions: if the section no longer exists, or does not have a process with specified logical process ID, the method generates KeyError exception.

Examples:

```
# print node names where the section processes are running
from FBS_API import FBSClient
fc=FBSClient()
si=fc.getSection('1234.MAIN') # get section information
for i in range(si.NProc):
      pi = si.getProcess(i+1) # logical PIDs run from 1 to NProc
      print pi.Node
```

isHeld

Purpose: returns 1 if the section is held, 0 otherwise. Section is considered held if:

- It is in state "waiting" and
- Its HoldTime is not None and it is in future or –1.

Synopsis: `isHeld()`

Arguments: none

Return value: Integer – 1 if the section is held, 0 otherwise

Exceptions: if the section no longer exists, the method generates KeyError exception.

```
Examples:
      # release all held sections of the job
      from FBS_API import FBSClient
      fc=FBSClient()
      j=fc.getJob('1234')
      for sn in j.sections():
            s = j.getSection(sn)
            if s.isHeld():    s.release()
```

hold

Purpose: holds the section either indefinitely, or until the specified time

Synopsis: `hold(hold_time = -1)`

Arguments: hold_time: String or Time – time after which the section should be released. The time can be specified in the following forms:

- string representation (see "Hold Time Specification" section)
- as value returned by time.localtime()
- -1 will hold the section indefinitely. This is default.

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the section no longer exists, the method generates KeyError exception.

Examples:

```
      # hold section by ID
      from FBS_API import *
      fc = FBSClient()
      si = fc.getSection(sys.argv[1])
      if len(argv) > 2:
            # user specified hold time
            si.hold(sys.argv[2])
      else:
            # hold forever
            si.hold()
```

release

Purpose: releases previously held section

Synopsis: `release()`

Arguments: none

Return value: 2-tuple (status, reason)
- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the section no longer exists, the method generates KeyError exception.

```
Examples:
      # release all held sections of the job
      from FBS_API import FBSClient
      fc=FBSClient()
      j=fc.getJob('1234')
      for sn in j.sections():
            s = j.getSection(sn)
            if s.isHeld():
                  sts, reason = s.release()
                  if not sts:
                        print 'Release %s: %s' % (sn, reason)
```

incPrio

Purpose: increases or decreases priority of a pending section. If the section is not pending, priority change does not have any effect.

Synopsis: `incPrio(increment)`

Arguments:
- increment: Integer – desired priority increment, or decrement if negative.

Return value: 2-tuple (status, reason)
- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Examples:

```
      # increment priority of all ready sections of a job
      from FBS_API import *
      fc=FBSClient()
      ji=fc.getJob('1234')
      for sn in ji.sections():
            si=ji.getSection()
            if si.State == 'ready': si.incPrio(1)
```

Exceptions: if the section no longer exists, the method generates KeyError exception.

kill

Purpose: sends request to kill the section

Synopsis: `kill(now=0)`

Arguments:

- now: Integer – optional argument, if now=1, SIGKILL signal will be sent to all running processes of the section, otherwise, or if unspecified, SIGINT will be sent, and later SIGKILL.

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Exceptions: if the section no longer exists, the method generates KeyError exception.

```
Examples:
      # kill immediately all sections of the job running on
      # particular node
      from FBS_API import *
      fc=FBSClient()
      ji=fc.getJob('1234')
      for sn in ji.sections():
            si=ji.getSection(sn)
            if si.State == 'running':
                  for i in range(si.NProc):
                        pi = si.getProcess(i+1)
                        if pi.State == 'running' and \
                                          pi.Node == 'pc123':
                              si.kill(1)        # kill it immediately
```

refresh

Purpose: update object's data members with current information

Synopsis: `refresh()`

Arguments: none

Return value: none

Exceptions: if the section no longer exists, the method generates KeyError exception.

Examples:

```
      # wait for section to complete and print its exit code
      from FBS_API import FBSClient
      fc=FBSClient()
      si=fc.getSection('1234.MAIN')
      while si.State != 'done':
            time.sleep(30)
            try:  si.refresh()
            except KeyError:
                  print 'Section exited with unknown exit code'
      print 'Exit code is ', si.ExitCode
```

10 FBSProcessInfo Class

This class is used to return information about the main batch process (the process which executes the command specified in job description) and the tree of its subprocesses. Objects of this class are returned by getProcess methods of FBSClient and FBSSectionInfo classes. API clients should not create objects of this class. That is why its constructor is left undocumented.

10.1 Data Members

FBSProcessInfo class has the following data members.

FBSProcessInfo Data Members

Name	Type	Description
BPID	String	FBS process id "jobid.sectname.procno"
UPID	Int	Unix process id
Node	String	Node name where the process is/was running
Status	String	Process state: initial, running, exited
Command	String	Command being executed
RsrcPoolDict	Dict[String]=String	Dictionary translating requested resource pool names into names of actually allocated resources
CPUTime(*)	Int	Accumulated CPU time of the main process
ACPUTime(*)	Int	Accumulated CPU time including all running and exited subprocesses
StartTime	Time	Process start time
ExitTime	Time	Exit time
ExitCode	Int	Process exit status
Core	0/1	Whether core was dumped
Signal	Int	Signal number if process was terminated by non-caught signal
Message(**)	String	Message sent with "fbs msg" command
Subprocesses(**)	List of FBSSubProcessInfo objects	List of objects representing subprocesses of the root process

Data members marked with (*) are more accurate for running processes if local_details parameter of the call to getProcess used to create this object, or refresh method of the object was set to 1.

Data members marked with (**) are returned only for running processes, and only if local_details parameter of the call to getProcess or refresh method was non-zero.

All data members of this class are “read-only” in the sense that any modifications of their value have no effect on actual batch process and will be ignored.

10.2 Methods

refresh

Purpose: update the object's data members with current information

Synopsis: `refresh(local_details=1)`

Arguments:

- local_details: Integer - optional argument, if local_details=1 (default), all available information about the process will be returned. Otherwise, if local_details=0, some returned information may be inaccurate, but the method will work faster. See description of FBSProcessInfo for details.

Return value: none

Exceptions: if the process no longer exists, the method generates KeyError exception.

```
Examples:
      # wait for a process to complete and print its exit code
      # and accumulated CPU time
      from FBS_API import FBSClient
      fc=FBSClient()
      pi=fc.getProcess('1234.MAIN.1')     # get process by batch PID
      while pi.Status != 'exited':
            time.sleep(30)
            try:  pi.refresh()
            except KeyError:
                  print 'Process disappeared'    # need to poll
                                                 # more often
                  break
      print 'Exit code is ', pi.ExitCode, ' CPU time ', pi.ACPUTime
```

11 FBSSubProcessInfo Class

FBSSubProcessInfo class is used to represent sub-processes of the root batch process. FBSProcessInfo object and referenced FBSSubProcessInfo objects represent tree of processes started by the root process.

11.1 Data Members

FBSSubProcessInfo class has the following data members:

FBSSubProcessInfo Data Members

Name	Type	Description
UPID	Int	Unix process ID
Command	String	Command being executed
CPUTime	Int	Accumulated CPU time for this process
ACPUTime	Int	Accumulated CPU time including subprocesses
Subprocesses	List of FBSSubprocessInfo Objects	List of objects representing subprocesses of this process

All data members of this class are "read-only" in the sense that any modifications of their value have no effect on actual batch process, and will be ignored.

12 FBSNodeClassInfo Class

This class represents information about a node class. Objects of this class are created by getNodeClass method of FBSClient class. API clients should not create objects of this class other than by calling this method. That is why constructor of this class is left undocumented.

12.1 Data Members

FBSNodeClassInfo has the following data members:

FBSNodeClassInfo Data Members

Name	Type	Description
Name	String	Name of the node class
Nodes	List of Strings	List of node names
ResourceCap	Dictionary	Resource capacity per node
LocalDisks	Dictionary	Resource name – to – root directory mapping for local scratch disks
Power	Floating point number	Relative power of the nodes of the class

All data members of this class are "read-only" in the sense that any modifications of their value have no effect on actual farm configuration, and will be ignored.

12.2 Methods

refresh

Purpose: updates object's data members with current information.

Synopsis: `refresh()`

Arguments: none

Return value: none

```
Examples:
      # periodically, print how many nodes of class Worker are down
      from FBS_API import *
      fc=FBSClient()
      ci = fc.getNodeClass('Worker')
      while 1:
            ci.refresh()      # in case configuration has changed
            ndown = 0
            for node in ci.Nodes:
                  if not fc.getNode(node).IsUp:
                        ndown = ndown + 1
            print time.ctime(time.time()), ndown
            time.sleep(60)
```

setRsrcCap

Purpose: sets resource capacity for all nodes of the class. This method should be used to introduce new local resources to the farm configuration. On success, this method updates ResourceCap data member with new values.

Synopsis: `setRsrcCap(rsrc_dict)`

Arguments:

- rsrc_dict: Dictionary – dictionary describing resource capacity for each node of the class. Resource names must be the dictionary keys, and capacity must be specified as dictionary values. For node attributes, values must be None. Resource names listed in this dictionary must be names of either new or existing local resources or node attributes. Global resource or resource pool names are not allowed.

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String – the job id on success, or textual explanation of the reason for failure

```
Example:
      # set resource capacity for a node class
      from FBS_API import *
      fc = FBSClient()
      nc = fc.getNodeClass('MCWorkers')
      # set capacity:
      #     cpu:  100 units
      #     disk: 18 units
      #     attributes: Linux, Worker, MC
      nc.setRsrcCap({'cpu':100,'disk':18,'Linux':None,
            'Worker':None, 'MC':None})
```

setLocalDisks

Purpose: sets new resource name – to – local scratch disk location mapping for nodes of the class. On success, this method updates LocalDisks data member with new values. After modifying the disk scratch mapping, administrator must re-start Launchers on all nodes of the class.

Synopsis: `setLocalDisks(disks)`

Arguments:

- disks: Dictionary – maps local resource names representing scratch disks on farm nodes of the class to the root directory of the scratch area.

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String – the job id on success, or textual explanation of the reason for failure

```
Example:
      # add new scratch disk on nodes of the class
      from FBS_API import *
      fc = FBSClient()
      nc = fc.getNodeClass('MCWorkers')
      dict = nc.LocalDisks
      dict['scratch2'] = '/stage2/scratch'
      nc.setLocalDisks(dict)
```

addNode

Purpose: adds a new node to the node class and to the farm. Initially, new nodes are created in held/down state. New nodes should be released later using appropriate methods of FBSClient or FBSNodeInfo classes.

Synopsis: `addNode(name)`

Arguments:

- name: String – name of new node. The node must not belong to any existing node class.

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure

- text: String – the job id on success, or textual explanation of the reason for failure

```
Example:
      # add 5 new nodes to 'Stream1' node class
      from FBS_API import FBSClient
      fc = FBSClient()
      nc = fc.getNodeClass('Stream1')
      for suffix in ['a','b','c','d','e']:
            name = 'fnpc%s' % suffix
            nc.addNode(name)
            fc.releaseNode(name)
```

removeNode

Purpose: removes a node from a node class. This operation can be performed only if no batch processes are running on the node. Combination of removeNode and addNode methods should be used to move a node from one node class to another.

Synopsis: `removeNode(name)`

Arguments:

- name: String – name of the node to remove

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String – the job id on success, or textual explanation of the reason for failure

```
Example:
      # move 5 nodes from one node class to another
      from FBS_API import FBSClient
      fc = FBSClient()
      nc1 = fc.getNodeClass('Stream1')
      nc2 = fc.getNodeClass('Stream2')
      for suffix in ['a','b','c','d','e']:
            name = 'fnpc%s' % suffix
            nc1.removeNode(name)
            nc2.addNode(name)
            fc.releaseNode(name)
```

13 FBSNodeInfo Class

This class represents information about individual farm node. Objects of this class are created by the getNode method of FBSClient class, and should not be created by API clients. That is why constructor of this class is undocumented.

13.1 Data Members

The class has the following data members:

FBSNodeInfo Data Members

Name	Type	Description
Name	String	Node name
Class	String	Node class
IsHeld	Int	Node is held (1) or not (0)
IsUp	Int	Node is up (1) or not (0)
HoldReason	String	Reason for being held
Resources	Dictionary dict[rsrc_name]= (usage, capacity)	Dictionary indexed by resource names with 2-tuples as values. In each tuple, first element is integer number representing current resource utilization, and second is resource capacity on empty node.
Processes	List of Strings	List of BPIDs of processes running on the node.

All data members of this class are "read-only" in the sense that any modifications of their value have no effect on actual farm configuration or state of the node, and will be ignored.

13.2 Methods

refresh

Purpose: updates object's data members with current information.

Synopsis: `refresh()`

Arguments: none

Return value: none

```
Examples:
      # periodically, print resource utilization statistics
      # for IO1 node
      from FBS_API import FBSClient
      fc=FBSClient()
      ni = fc.getNode("IO1")
      while 1:
            ni.refresh()
            if not ni.IsUp:
                  print '--Node is down'
            elif ni.IsHeld:
                  print '--Node is held: ', ni.HoldReason
            else:
                  for rn in ni.Resources.keys():
                        usage, cap = ni.Resources[rn]
                        print 'Resource %s: used %d out of %d' %\
                                    (rn, usage, cap)
            time.sleep(60)
```

14 FBSQueueInfo Class

This class represents information about an individual FBSNG queue. Objects of this class are created by the getQueue method of FBSClient class, and should not be created by API clients. That is why constructor of this class is undocumented.

14.1 Data Members

This class has the following data members:

FBSQueueInfo Data Members

Name	Type	Description
Name	String	Queue name
DefProcType	String	Default process type
IsLocked	Int	1 if the queue is locked, 0 if not
IsHeld	Int	1 if the queue is held, 0 if not
Sections	List of Strings	List of section ids ordered according to scheduling algorithm. First section will be scheduled first.
SectState	Dict[sectid]= String	Section state dictionary indexed by section id.
SectPrio	Dict[sectid]= Int	Section priority dictionary indexed by section id.
Npending	Int	Number of pending (waiting or ready) sections in the queue.
Nrunning	Int	Number of running sections in the queue.
MaxSPrio	Int	Maximum section priority for this queue.
SPGap	Int	Section priority gap.
MaxQPrio	Int	Maximum priority of this queue.
MinQPrio	Int	Minimum priority of this queue.
Prio	Int	Current priority of this queue.
QPDec	Int	Queue priority decrement.
QPGap	Int	Queue priority gap for this queue.
Users	List of Strings	List of users allowed to use the queue.
ProcTypes	List of Strings	List of process types allowed in the queue.
CPUTimeLimit	Int	Process CPU time limit in seconds or –1 if no limit
RealTimeLimit	Int	Process elapsed time limit in seconds or –1 if no limit

All data members of this class are "read-only" in the sense that any modifications of their value have no effect on actual farm configuration or state of the queue, and will be ignored until update method is called.

14.2 Methods

hold

Purpose: holds the queue temporarily preventing new sections waiting in this queue from starting.

Synopsis: `hold()`

Arguments: none

Return value: 2-tuple (status, reason)
- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

```
Examples:
      # hold all FBSNG queues
      from FBS_API import FBSClient
      fc=FBSClient()
      for qn in fc.getQueueList():
            q = fc.getQueue(qn)
            sts, reason = q.hold()
            if not sts:
                  print 'Can not hold queue <%s>: %s' % (qn, reason)
```

release

Purpose: releases previously held queue

Synopsis: `release()`

Arguments: none

Return value: 2-tuple (status, reason)
- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

Examples:
```
      # release all empty queues
      from FBS_API import *
      fc=FBSClient()
      for qn in fc.getQueueList():
            q = fc.getQueue(qn)
            if q.Sections:
                  # queue is not empty: print list of sections
                  # and skip it
                  print 'Queue <%s> is not empty: ' % qn, \
                        string.join(q.Sections)
                  continue
            sts, reason = q.release()
            if not sts:
                  print 'Can not release queue <%s>: %s' % (qn, reason)
```

lock

Purpose: lock the queue: disable new sections submission into this queue

Synopsis: `lock()`

Arguments: none

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

```
Examples:
      # lock all queues with default process type "Worker"
      from FBS_API import FBSClient
      fc=FBSClient()
      for qn in fc.getQueueList():
            q = fc.getQueue(qn)
            if q.DefProcType == 'Worker':
                  sts, reason = q.lock()
                  if not sts:
                        print 'Can not lock queue <%s>: %s' % \
                              (qn, reason)
```

unlock

Purpose: unlock the queue

Synopsis: `unlock()`

Arguments: none

Return value: 2-tuple (status, reason)

- status: Integer – is 1 on success and 0 on failure
- reason: String – on failure, textual explanation of the reason for failure

```
Examples:
      # unlock all queues and make sure they are not held
      from FBS_API import *
      fc=FBSClient()
      for qn in fc.getQueueList():
            q = fc.getQueue(qn)
            sts, reason = q.unlock()
            if not sts:
                  print 'Can not unlock <%s>: %s' % (qn, reason)
                  continue
            if q.IsHeld:
                  q.release()
```

update

Purpose: updates queue parameters with values of the following object's data members:

- QPGap
- QPInc
- QPDec
- MaxQPrio
- MinQPrio
- MaxSPrio (u)
- MinSPrio (u)
- SPGap (u)
- Prio
- DefProcType

- RealTimeLimit
- CPUTimeLimit
- Users
- ProcTypes

On success, the method updates the object's data members with new values. Fields marked with (u) can be modified by an authorized user (listed in Users). Other fields can be modified only by FBSNG administrator.

Synopsis: `update()`

Arguments: none

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String –textual explanation of the reason for failure

```
Example:
      # create new queue and set its parameters,
      # then unlock the queue
      from FBS_API import *
      fc = FBSClient()
      sts, reason = fc.createQueue('LongQ')
      if not sts:
            print 'Can not create queue: ', reason
      else:
            qi = fc.getQueue('LongQ')
            qi.QPDec = 5
            qi.MaxQPrio = 1000
            qi.QPGap = 15
            qi.CPUTimeLimit = None  # unlimited
            qi.MaxSPrio = 100
            qi.update()
            qi.unlock()
            print 'Done.'
```

refresh

Purpose: updates object's data members with current information.

Synopsis: `refresh()`

Arguments: none

Return value: none

```
Examples:
      # drain a queue:
      # lock FastQ queue then wait until it's empty.
      from FBS_API import FBSClient
      fc=FBSClient()
      q = fc.getQueue('FastQ')
      q.lock()
      while q.Sections:
            # stil not empty, print numbers, then
            # sleep for a minute and check again
            print 'pending: %d, running: %d, total: %d' %\
                  (q.NPending, q.NRunning, len(q.Sections))
            time.sleep(60)
            q.refresh()
      # now it is empty.
```

15 FBSProcTypeInfo Class

The FBSProcTypeInfo class is used to represent configuration and resource utilization information for FBSNG process types. Objects of this class are created by the getProcessType method of FBSClient class, and should not be created by API clients. That is why constructor of this class is undocumented.

15.1 Data Members

FBSProcTypeInfo class has the following data members:

FBSProcTypeInfo Data Members

Name	Type	Description
Name	String	Name of the process type
MaxPrioInc	Int	Maximum allowed priority increment or None
RsrcQuota	Dict	Process type quota for resource usage
RsrcUsage	Dict	Amount of resources currently used by processes of this type
ProcRsrsDefaults	Dict	Default resource utilization per process
SectRsrsDefaults	Dict	Default resource utilization per section
Users	List of Strings	List of users allowed to use the process type
NodesAllow	List of Strings	List of nodes processes of this type are allowed to run on
NodesDisallow	List of Strings	List of nodes processes of this type are disallowed to run on
MaxNodeCount	Int or None	Maximum number of different nodes the processes of this type are allowed to run on at any time. None means no limit.
CurrentNodeCount	Int	Number of nodes processes of the type are currently running on
RealTimeLimit	Int	Maximum allowed process elapsed time in seconds. –1 means no limit.
CPUTimeLimit	Int	Maximum allowed process CPU time in seconds. –1 means no limit.
MinNice	Int	Minimal NICE factor for processes of this type

All data members of this class are "read-only" in the sense that any modifications of their value have no effect on actual farm configuration or state of the queue, and will be ignored.

Users data member is contains list of users allowed to use the process type. If the list contains "*" string, the rest of the list is ignored, and all users are permitted to use the process type.

NodesAllow and NodesDisallow members can be used by authorized users (those listed in Users) or FBSNG administrator to control which nodes can be chosen to start batch processes on. If any of the two lists contains wild card string "*", the rest of the list is ignored, and all the nodes are considered as included in the list. FBSNG uses the following algorithm to decide whether a process of the process type can start on a particular node:

- If NodesDisallow contains "*", then the node is allowed only if NodesAllow contains the name of the node

- Else, if NodesAllow contains "*", then the node is allowed only if it is not listed in NodesDisallow
- Else, if neither list contains "*", the node is allowed if it is listed in NodesAllow but not in NodesDisallow

As you can see, ambiguities like both lists contain "*", or the name of the node is present in both list are always resolved by not allowing the node.

Note that this mechanism *does not* supersede FBSNG node selection algorithms. NodesAllowe and NodesDisallow can be used only in addition to regular resource management algorithms described in FBSNG Resources document.

15.2 Methods

setSectRsrcDefs

Purpose: sets new default section resource requirements

Synopsis: `setSectRsrcDefs(rsrc_dict)`

Arguments:

- rsrc_dict: Dictionary - dictionary describing section default resource requirements for sections of the process type. Resource names must be the dictionary keys, and the requirements must be specified as dictionary values.

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String – the job id on success, or textual explanation of the reason for failure

```
Example:
      # create new process type and specify quotas and default
      # resource requirements and maximum allowed priority
      # increment
      from FBS_API import FBSClient
      fc = FBSClient()
      sts, reason = fc.createProcessType('NewWorker')
      if not sts:
            print 'Can not create process type: ', reason
      else:
            pt = fc.getProcessType('NewWorker')
            pt.setProcRsrcDefs({'cpu':100,'Linux':None})
            pt.setSectRsrcDefs({'nfs_disk':3})
            pt.setRsrcQuota({'cpu':10000,'nfs_disk':21})
            pt.setMaxPrioInc(10)
```

setProcRsrcDefs

Purpose: sets new default process resource requirements.

Synopsis: `setProcRsrcDefs(rsrc_dict)`

Arguments:

- rsrc_dict: Dictionary - dictionary describing process default resource requirements for the process type. Resource names must be the dictionary keys, and the requirements must be specified as dictionary values. For node attributes, value must be None.

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String – the job id on success, or textual explanation of the reason for failure

```
Example: (see the example for setSectRsrcDefs)
```

setRsrcQuota

Purpose: sets new resource utilization quota for the process type

Synopsis: `setRsrcQuota(rsrc_dict)`

Arguments:

- rsrc_dict: Dictionary - dictionary describing resource allocation quotas for the process type. Resource names must be the dictionary keys, and the requirements must be specified as dictionary values. If a resource is not present in the dictionary, the process type will be able to use unlimited amount of the resource.

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String –textual explanation of the reason for failure

```
Example: (see the example for setSectRsrcDefs)
```

setMaxPrioInc

Purpose: sets new maximum priority increment for sections of the process type.

Synopsis: `setMaxPrioInc(new_max)`

Arguments:

- new_max: Integer or None – new value of the maximum. None allows unlimited priority increments.

Return value: 2-tuple (status, text)

- status: Integer – is 1 on success and 0 on failure
- text: String –textual explanation of the reason for failure

```
Example: (see the example for setSectRsrcDefs)
```

setMaxNodeCount

Purpose: sets new maximum number of nodes processes of the type are allowed to run on.

Synopsis: `setMaxNodeCount(new_max)`

Arguments:
- new_max: Integer or None – new value of the maximum. None means no limit.

Return value: 2-tuple (status, text)
- status: Integer – is 1 on success and 0 on failure
- text: String –textual explanation of the reason for failure

setCPUTimeLimitt

Purpose: sets new process CPU time limit.

Synopsis: `setCPUTimeLimit(new_max)`

Arguments:
- new_max: Integer – new CPU time limit in seconds. -1 means no limit.

Return value: 2-tuple (status, text)
- status: Integer – is 1 on success and 0 on failure
- text: String –textual explanation of the reason for failure

setRealTimeLimitt

Purpose: sets new process elapsed time limit.

Synopsis: `setRealTimeLimit(new_max)`

Arguments:
- new_max: Integer – new elapsed time limit in seconds. -1 means no limit.

Return value: 2-tuple (status, text)
- status: Integer – is 1 on success and 0 on failure
- text: String –textual explanation of the reason for failure

refresh

Purpose: updates object's data members with current information.

Synopsis: `refresh()`

Arguments: none

Return value: none

```
Examples:
      # monitor resource utilization per process type
      from FBS_API import FBSClient
      fc=FBSClient()
      types = fc.getProcessTypeList()

      # get list of all resources
      rlist = fc.getLocalRsrcList() + fc.getGlobalRsrcList()

      # make list of FBSProcTypeInfo objects
      ptlist = []
      for pt_name in types:
            ptlist.append(fc.getProcessType(pt_name))

      while 1:
            for pti in ptlist:
                  pti.refresh()
                  print 'Process type ',pti.Name,':'
                  for rn in rlist:

                        if not pti.RsrcQuota.has_key(rn) or \
                              pti.RsrcQuota[rn] == None:
                              # no quota defined = unlimited
                              quota = '(unlimited)'
                        else:
                              quota = pti.RsrcQuota[rn]

                        if not pti.RsrcUsage.has_key(rn):
                              # not used by this proc. class
                              usage = 0
                        else:
                              usage = pti.RsrcUsage[rn]

                        print 'Resource %s: %s used out of %s quota' %\
                              (rn, usage, quota)
```

setUsers

Purpose: to define list of users authorized to use the process type (see Users data member). Only FBSNG administrator can use this method.

Synopsis: setUsers(list_of_users)

Arguments:

- list_of_users: List of Strings – list of users authorized to use the process type. Value of this argument supersedes old value. If the list contains string "*", the rest of the list is ignored.

```
Example:
      #
      # Replace user "bob" with user "alice" in all process type
      # user lists
      #

      from FBS_API import FBSClient

      fc=FBSClient()
      for ptn in fc.getProcessTypeList():
            pti = fc.getProcessType(ptn)
            users = pti.Users[:]
            if "bob" in users:
                  users.remove("bob")
                  users.append("alice")
                  pti.setUsers(users)
```

allowNodes

Purpose: defines list of nodes the processes of this type are allowed to run on (see NodesAllow data member). This method can be used only by FBSNG administrator or an user authorized to use the process type.

Synopsis: allowNodes(list_of_nodes)

Arguments: list_of_nodes: List of Strings – list of node named where processes of this type are allowed to start. Value of this argument supersedes old value. If the list contains string "*", the rest of the list is ignored.

disallowNodes

Purpose: defines list of nodes the processes of this type are disallowed to run on (see NodesDisllow data member). This method can be used only by FBSNG administrator or an user authorized to use the process type.

Synopsis: disallowNodes(list_of_nodes)

Arguments: list_of_nodes: List of Strings – list of node named where processes of this type are not allowed to start. Value of this argument supersedes old value. If the list contains string "*", the rest of the list is ignored.

```
Example 1:
      #
      # Something is wrong with node "pc13". Make sure processes
      # of type "Worker" will not run there. For processes of
      # this type, this will be equivalent to holding the node.
      #

      from FBS_API import FBSClient

      fc=FBSClient()
      pti=fc.getProcessType("Worker")
      pti.allowNodes(["*"])
      pti.disallowNodes(pti.NodesDisallow + ["pc13"])

Example 2:
      #
      # Run processes of type "dbwriter" only on nodes
      # pc1 - pc15
      #

      from FBS_API import FBSClient

      fc=FBSClient()
      pti=fc.getProcessType("Worker")
      lst = []
      for i in range(15):
            lst.append("pc%d" % (i+1))
      pti.disallowNodes(["*"])
      pti.allowNodes(lst)
```

16 FBSEventListener Class

FBSEventListener is a virtual base class designed to provide interface to FBSNG Event Manager. Client application is supposed to create a subclass of this class to implement its own asynchronous event receiving and processing functionality.

16.1 Methods

The following are methods of FBSEventListener class available to the client application.

subscribe

Purpose: subscribe notifications about a section status changes. This method can be called more than once to subscribe to the notifications about more than one section of one or more jobs.

Synopsis: subscribe(sect_id)

Arguments:

- sect_id: String – ID of the section.

Return value: none

Generates KeyError exception if the section does not exist.

unsubscribe

Purpose: stop receiving notification about a section status changes.

Synopsis: unsubscribe(sect_id = None)

Arguments:

- sect_id: String – ID of the section. If omitted, will cancel all current subscriptions.

Return value: none

sectionState

Purpose: return most recent state of the section as received as received during latest call to *subscribe* or *wait* method.

Synopsis: sectionState(sect_id)

Arguments:

- sect_id: String – ID of the section.

Return value: String – state of the section: “waiting”, ”ready”, ”running”, etc.

Generates KeyError exception if the EventListener is not subscribed to notifications about the specified section.

processState

Purpose: returns most recent state of the batch process as received during latest call to *subscribe* or *wait* method.

Synopsis: processState(sect_id, procno)

Arguments:
- sect_id: String – ID of the section.
- Procno: Integer – Logical process ID

Return value: String – state of the batch process: "pending", "running", "exited".

Generates KeyError exception if the EventListener is not subscribed to notifications about the specified section.

wait

Purpose: wait for state change notifications. When this method is called, EventListener object receives state change notification events from FBSNG. While the call to this method is in progress, EventListener calls its own virtual methods sectionStateChanged, processStateChanged and sectionDeleted described below. User can control how long to wait for events by specifying optional nmax or tmo or both arguments.

When one of the sections the EventListener is subscribed to gets deleted, EventListener automatically un-subscribes from the notifications about the section.

Synopsis: wait(nmax=None, tmo=-1)

Arguments:
- nmax: Integer – if specified, the method will block until the specified number of events are received. The number of received events includes ignored events too. If not specified, the method will wait forever, or until the time specified by "tmo" argument elapses.
- tmo: Integer – if specified, the method will block until the specified number of seconds elapses. If not specified, the method will wait forever, or until the number of events specified by "nmax" argument is received.

If both arguments are specified, the method will block until either the time-out passes, or certain number of events are received, whichever occurs earlier.
If neither argument is specified, the method will block forever.

Return value: Integer – number of events received.

16.2 Virtual Methods

FBSEventListener has three virtual methods that are to be overridden by the user to add necessary event processing functionality. User does not have to override all three of them. Methods of the base class do nothing, so if they are not overridden, corresponding events will be ignored.

sectionStateChanged

Purpose: this method is called to notify that the state of one of sections the Event Listener object is subscribed to has changed.

Synopsis: sectionStateCanged(self, sect_id, old_state, new_state)

Arguments:
- sect_id: String – ID of the section.
- old_state: String – previous state of the section
- new_state: String – new state of the section

Return value: ignored

processStateChanged

Purpose: this method is called to notify that the state of one of processes of a section the Event Listener object is subscribed to has changed.

Synopsis: processStateCanged(self, sect_id, procno, old_state, new_state)

Arguments:
- sect_id: String – ID of the section
- procno: Integer – logical ID of the process
- old_state: String – previous state of the section
- new_state: String – new state of the section

Return value: ignored

sectionDeleted

Purpose: this method is called when one of sections the Event Listener object is subscribed has been deleted from FBSNG. The user does not have to explicitly un-subscribe from the deleted section information because EventListener does it automatically.

Synopsis: sectionDeleted(self, sect_id)

Arguments:
- sect_id: String – ID of the section.

Return value: ignored

16.3 Event Listener Programming Examples

Example 1: Monitoring section status

The following example illustrates features of the Event Listener API. This is sample application, which can be used to wait for one or more job sections to finish.

```
from FBS_API import FBSEventListener

# define my own event listener class
class MyEventListener(FBSEventListener):
      def __init__(self, sid_list):

            # call the superclass constructor
            FBSEventListener.__init__(self)

            # subscribe to all sections
            self.Nalive = 0   # number of not-yet-deleted
                              # sections
            for sid in sid_list:
                  try:  self.subscribe(sid)
                  except KeyError:
                        print 'Section does not exist: %s' % sid
                  else:
                        self.Nalive = self.Nalive + 1

      # We are interested in section state changes
      # Override base class methods

      def sectionStateChanged(self, sid, old_state, new_state):
            print 'Section %s: %s -> %s' % \
                  (sid, old_state, new_state)

      def sectionDeleted(self, sid):
            print 'Section %s deleted' % sid
            self.Nalive = self.Nalive - 1

      # We are not concerned about individual process states,
      # therefore, we will leave processStateChanged method
      # as it is.

#
# This is our main code
# Usage: python wait.py <sid> …
#
import sys
import time

listener = MyEventListener(sys.argv[1:])
while listener.Nalive > 0:
      nevents = listener.wait(nmax = 1, tmo = 10)
      print '%s: %d events received' %\
            (time.ctime(time.time()), nevents)
print 'All sections are deleted'
```

Example 2: Polling for section status changes

Although EventListener provides necessary functionality to avoid using polling technique to monitor batch job status, it may be worthwhile to illustrate how EventListener can be used in that mode. The following example works in the way very similar to calling FBSSectionInfo.refresh method periodically and printing state of the section when it changes.

```
from FBS_API import FBSEventListener
import sys

l=FBSEventListener()
try:  l.subscribe('1234.Main')
except KeyError:
      print 'Section does not exist'
      sys.exit(1)
old_state = l.sectionState('1234.Main')
print old_state
deleted = 0
while not deleted:
      # wait for 1 event or 10 seconds
      # remove second argument to block until next event
      n = l.wait(1, 10)
      if not n:
            # nothing happened
            continue
      try:  new_state = l.sectionState('1234.Main')
      except KeyError:
            # section deleted
            deleted = 1
      else:
            if new_state != old_state:
                  print new_state
                  old_state = new_state
print 'Section has been deleted'
```

Appendix A: JDF Format

Job Description File (JDF) is a plain text file that contains complete description of an FBSNG job to be submitted. JDF consists of one or more section descriptions. Each section description consists of SECTION line followed by one or more section parameter lines:

```
SECTION <section-name>
<parameter> = <value>
<parameter> = <value>
…
SECTION <section-name>
<parameter> = <value>
<parameter> = <value>
…
```

Blank lines in JDF are ignored. Pound character (#) in the beginning or in the middle of line can be used to indicate start of a comment. Blank characters around required "equals" signs (=) are ignored. Order in which sections appear in JDF is not important. Each section must have a unique name. Only alphanumeric characters, underscore (_) and hyphen (-) may be used for section names. Section names must start with a letter. Section names are case-sensitive. "SECTION" keyword and parameter names must be in upper case. If the same parameter is appears more than once in the same section description, the value set by the last occurrence will be used, otherwise section parameters can be listed in any order.

For each section, QUEUE and EXEC parameters must be specified. All other parameter values have defaults listed in the table below.

FBSNG API recognizes the following section parameters:

JDF Fields

Keyword (synonym)	**Type (Default)**	**Corresponding FBSSectionDesc Field**	**Description**
NUMPROC	Integer (1)	NProc	Number of processes in the section
QUEUE	string (required)	Queue	Queue name
EXEC	string (required)	Exec	Command to execute
PROC_TYPE	string (Queue default)	ProcType	Section process type
NEED	0/1 (0)	Need	NEED flag
LEADER_ONLY	0/1 (0)	LeaderOnly	LEADER_ONLY flag
PRIO_INC	integer (0)	PrioInc	Initial section priority increase
NICE	integer (0)	Nice	Nice level for processes

Keyword (synonym)	**Type (Default)**	**Corresponding FBSSectionDesc Field**	**Description**
HOLD_TIME (AFTER)	date/time (none)	HoldTime	Time to hold the section
PROC_RESOURCES	<rsrc>:<n> … <attr> … (none)	PerProcRsrc	Resources required for each process in addition to process type default
SECT_RESOURCES	<rsrc>:<n> … (none)	PerSectRsrc	Resources required for the section in addition to process type default
MAILTO	e-mail address (none)	MailTo	Address to send section report to.
DEPEND	dep. Expression (none)	Depend	Section dependency specification
STDERR	pattern (FBS_%j.%n.err)	Stderr	Template for process stderr file name
STDOUT	pattern (FBS_%j.%n.out)	Stdout	Template for process stdout file name
SECT_STDOUT	pattern (none)	SectOutput	Template for section log file name
ON_NODES	<node name> … (no restrictions)	OnNodes	List of nodes processes of the section are allowed to start on

For more information about meaning of section parameters see FBSNG User's Guide.

Example of JDF:

```
SECTION Pre-stage
QUEUE = PreStageQ                    # QUEUE is required
# NUMPROC is 1 by default
EXEC = /home/e1234/stage-data.sh -v XYZ123
                                     # EXEC is required

#-------------------------------------------------
# Main production section
SECTION Process
QUEUE = E1234-ProdQueue
NUMPROC = 10                         # run 10 processes
DEPEND = done(Pre-stage)             # start only if Pre-stage
                                     # was successful
PROC_RESOURCES = Linux scratch:5     # run only on Linux computers
                                     # request 5 GB of scratch
                                     # disk space
EXEC = process-data.sh -v XYZ123     # relative path (w.r.t. HOME)
STDOUT = logs/%j.%n.out              # template for process stdout

#-------------------------------------------------
# Next section will store output to tape
SECTION Save_data
EXEC = save-data.sh -v OUT567
DEPEND = done(Process)
QUEUE = IOQueue

#-------------------------------------------------
SECTION Clean_up        # this is emergency clean-up section.
                        # run it only if something went wrong

QUEUE = FastQ
PRIO_INC = 15                  # submit at higher priority
EXEC = clean-up.sh -i XYZ123 -o OUT567
DEPEND = failed(Process) || failed(Pre-stage)
STDERR = /dev/null             # ignore stderr
STDOUT = /dev/null             # and stdout
MAILTO = err-log               # mail section output
```

Appendix B: Hold Time Representation

When submitting a job, the user can request that one or more sections of the job should be held until some time in future in the queue without starting. The time to hold the section until is called the hold time. Hold time can be specified in the JDF or in the HoldTime field of an FBSSectionDesc object. In both places, time can be specified as a text string in one of the following formats:

[<date>-][<time>] - absolute date/time specification
Use this format for a specific point of time on specific date. If the date field is omitted, today is assumed. If time is omitted, current time of day is assumed.

+<days>-[<time>] - absolute time specification at later day
This form represents a specific point of time today or <days> days later. If time is omitted, midnight is assumed. If <days> is 0, it is interpreted as "the next time the wall clock shows the specified time". This can be today or tomorrow. <days> is 1 means "tomorrow at the specified time", <days>=2 means "the day after tomorrow at the specified time" and so on.

+<time> - relative time specification
This form specifies future time as current day/time plus a certain number of hours, minutes and seconds.

In addition, hold time can be specified as the keyword “forever” (hold section indefinitely) or “None” (do not hold, start as soon as possible).

In the above formats, <date> is [mm/]dd[/yyyy]. If month (mm) is omitted, the current month is implied, and if year (yyyy) is omitted, the current year is implied.

<time> is hh[:mm[:ss]]. If minutes field (mm) is omitted, 00 is implied. If seconds field (ss) is omitted, 00 is implied.

<days> is an unsigned integer number.

Examples of hold time representation:

"4/1/2000-"	hold until midnight April 1 2000. If that is already in the past, do not hold.
"7/8/2000-05:00"	hold until 5am July 8’Th 2000
"10:00:00"	hold until 10am today. If it's past 10am, do not hold.
"+3:00:00"	hold for 3 hours
"+1-06:00:00"	hold until tomorrow 6am
"+0-12:00:00"	if current time is before noon, then hold until noon today, otherwise until noon tomorrow.
"forever"	hold the section forever
"None"	do not hold at all

Appendix C: More API Examples

The following example to illustrates job submission and monitoring using FBSClient, FBSJobDesc, FBSSectionDesc, FBSSectionInfo and FBSProcessInfo classes:

```
#
# Name: exec.py
#
# This is a script which can be used to emulate LSF-style
# single-process job submission.
#
# Usage: python exec.py
#      -q <queue> [-v] [-n <numproc>] [-p <proctype>]
#      [-w <seconds or -1>] [-h <hold-date-time>]
#      [-r <resources>] [-i <priority increment>]
#      [-s <section-name>]
#      <command> [<argument> ...]
#
# exec.py submits single-section FBSNG job that
# will execute the specified command. If requested,
# exec.py will wait for the section to complete and exit with
# section exit code. User can specify additional parameters
# such as process type, hold time, extra resources and
# FBSNG queue priority increment.
# If requested, the script will print section parameters as
# JDF and report where and when individual processes start.
#

from FBS_API import *
import string
import getopt
import sys
import Parser
import time

usage = """
exec.py -q <queue> [-v] [-n <numproc>] [-p <proctype>]
      [-w <seconds or -1>] [-h <hold-date-time>]
      [-r <resources>] [-i <priority increment>]
[-s <section-name>]
<command> [<argument> ...]
"""

def main():
      over_dict = {}
      sn = "Exec"       # default section name
      wait = 0          # by default, do not wait
      verbose = 0
      queue = None

      try:  opts, args = getopt.getopt(sys.argv[1:],
            'vq:p:n:w:s:r:a:i:')
      except getopt.error, msg:
            print msg
            return
      for opt, val in opts:
            if opt == '-q':         queue = val
            elif opt == '-v': verbose = 1
            elif opt == '-h': over_dict["HoldTime"] = val
            elif opt == '-p': over_dict["ProcType"] = val
            elif opt == '-s': sn = val
            elif opt == '-r':       over_dict["PerProcRsrc"] = \
```

```
                  Parser.wordsToDict(val, defValue = 0)
      elif opt == '-n':
            over_dict["NProc"] = string.atoi(val)
      elif opt == '-i':
            over_dict["PrioInc"] = string.atoi(val)
      elif opt == '-w':
            if val == 'forever':    wait = -1
            else:       wait = string.atoi(val)

if not queue:
      # queue is required
      print usage
      return 1

cmd = string.join(args)
if not cmd:
      print usage
      return 1

fc=FBSClient()          # establish connection to FBSNG
j=FBSJobDesc()          # create job description

# create section description, set section parameters
s=FBSSectionDesc(sn, override = over_dict,
Queue = queue, Exec = cmd)

if verbose:
      print s     # print as JDF

# add the section to the job and submit the job
j.addSection(s)
sts, jid = fc.submitJob(j)
if not sts:
      # error
      print 'Submit failed: %s' % jid
      return 1
else:
      # job submitted, print its id
      print 'Job %s' % jid

# if requested, wait for completion
t = time.time()
sid = '%s.%s' % (jid, sn)
s = fc.getSection(sid)
state = ''
print_start = 1
while wait < 0 or time.time() < t + wait:
      try:  s.refresh()
      except KeyError:
            # section disappeared. See history.
            print 'not found'
            return -1
      if verbose and s.State != state:
            # report state change
            state = s.State
            msg = '%s' % state
            t = time.time()
```

```
                  if state == 'running':
                        t = s.StartTime
                        try:
                              # print where 1-st process started
                              pi=s.getProcess(1)
                              msg = msg + (' on %s, pid = %d' %
                                    (pi.Node, pi.UPID))
                        except:
                              # process not found (?)
                              pass
                        print_start = 0
                  elif state == 'done':
                        if print_start:
                              print '%s: running' % \
                                    time.ctime(s.StartTime)
                              print_start = 0
                        t = s.EndTime
                  print '%s:%s' % (time.ctime(t),msg)
            if s.State in ['waiting','ready','running']:
                  # it is not over yet, wait
                  time.sleep(10)
                  continue
            # now the section is done.
            print s.State, s.ExitCode
            return s.ExitCode

if __name__ == '__main__':
      sys.exit(main())
```

The following example illustrates features of FBSQueueInfo and FBSSectionInfo classes and how to use them to monitor state of FBSNG queues.

```
#
# Name: getq.py
#
# Print information about queues and sections in the queues
#
# Usage: python getq.py (-1|[-s])
#
# Options:  -1    - print 1-line summary per queue, do not
#                 print individual sections
#           -s    - short and quick output: print limited
#                 information about sections
#

from FBS_API import FBSClient
import time
import sys
import getopt

short = 0
one_line = 0

opts, args = getopt.getopt(sys.argv[1:],'s1')
for opt, val in opts:
      if opt == '-s': short = 1
      if opt == '-1': one_line = 1

fc = FBSClient()

ql = fc.getQueueList()
for qn in ql:
      q = fc.getQueue(qn)
      print 'Queue %-10.10s: Prio=%-3d +%-3d -%-3d Gap=%-3d' % (
            qn, q.Prio, q.QPInc, q.QPDec, q.QPGap),
      print 'R/P/T=%-3d/%-3d/%-3d' % (q.NRunning, q.NPending,
            len(q.Sections))
      if not one_line:
            if short:
                  for sid in q.Sections:
                        print '%-15.15s(%3.3s)   Prio=%3d' % (
                              sid, q.SectState[sid], q.SectPrio[sid])
            else:
                  for sid in q.Sections:
                        try:  s = fc.getSection(sid)
                        except: continue
                        print '%-15.15s(%3.3s) %10s %-3d %1d %3d %3d'%\
                              (sid, s.State[:3], s.ProcType, s.NProc,
                                    s.Need, s.Prio, s.QIndex)
                        if s.HoldTime:
                              print  '  Hold until: %s' %
                                    time.ctime(s.HoldTime)
                        if s.Depend:
                              print '  Depends: %s' % s.Depend
```

Appendix D: Glossary of Terms

Job ID - A string that uniquely identifies an FBSNG Job. A job is assigned an ID at the submission time.

Section ID – A string that uniquely identifies an FBSNG job section. Section ID is constructed from job ID and section name:

```
<SectionID> = <JobID>.<SectionName>
```

For example, section named "Main" of job with job ID "1234" will have section ID "1234.Main".

Batch Process ID (BPID) – A string that uniquely identifies a batch process. It is constructed from section ID and logical process ID:

```
<BPID> = <SectionID>.<LogicalPID>
```

For example, process number 3 of section with ID "1234.Main" will have BPID "1234.Main.3".

Logical Process ID – Sequence number of the processes of the section. FBSNG job sections can be viewed as arrays of identical processes. Logical process ID is an index in such an array. Logical process IDs of a section processes starts from 1.

Main or Root Batch Process - The batch process actually started by FBSNG. It is referred as "root" because it usually is the root of a UNIX process tree.

Index